Finding Most Popular Indoor Semantic Locations Using Uncertain Mobility Data

Li, Huan; Lu, Hua; Shou, Lidan; Chen, Gang; Chen, Ke

Published in:
IEEE Transactions on Knowledge and Data Engineering

DOI (link to publication from Publisher):
10.1109/TKDE.2018.2875096

Publication date:
2019

Document Version
Accepted author manuscript, peer reviewed version

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2875096, IEEE Transactions on Knowledge and Data Engineering

Finding Most Popular Indoor Semantic Locations Using Uncertain Mobility Data

Huan Li, Hua Lu, *Senior Member, IEEE,* Lidan Shou, Gang Chen, and Ke Chen

Abstract—Knowing popular indoor locations can benefit many applications like exhibition planning and location-based advertising, among others. In this work, we use uncertain historical indoor mobility data to find the top-k popular indoor semantic locations with the highest flow values. In the data we use, an object positioning report contains a set of samples, each consisting of an indoor location and a corresponding probability. The problem is challenging due to the difficulty in obtaining reliable flow values and the heavy computational workload on probabilistic samples for large numbers of objects. To address the first challenge, we propose an indoor flow definition that takes into account both data uncertainty and indoor topology. To efficiently compute flows for individual indoor semantic locations, we design data structures for facilitating accessing the relevant data, a data reduction method that reduces the intermediate data to process, and an overall flow computing algorithm. Furthermore, we design search algorithms for finding the top-k popular indoor semantic locations. All proposals are evaluated extensively on real and synthetic data. The evaluation results show that our data reduction method significantly reduces the data volume in computing, our search algorithms are efficient and scalable, and the top-k popular semantic locations returned are in good accord with ground truth.

Index Terms—Indoor space, Indoor mobility data, Indoor flows

1 Introduction

The past few years have witnessed the rapid deployment of multiple indoor location-based service infrastructures [1] as well as the high penetration of powerful smartphones [2]. On the other side, people (most are smartphone users) spend significant part (up to 87%) of their daily life indoors as disclosed by multiple studies [16], [22]. Driven jointly by these key factors, people's indoor movements are increasingly datafied, which produces large volumes of indoor mobility data. Such data takes different formats [24] depending on the particular underlying indoor positioning technology [10], [25], [41].

Akin to what has been done using outdoor GPS data [8], [30], [33], proper analysis on indoor mobility data can reveal insights that are otherwise difficult or even impossible to obtain. As a typical example, by analyzing the historical mobility data we can determine the number of people passing by a particular indoor region during a past time interval, which can be useful in many application scenarios such as location-based advertising [12] and environmental quality improvement [29]. We use *indoor flow* to refer to such findings.

In this paper, we formulate and study the problem of finding the top-k *indoor semantic locations* with the highest indoor flows during a past time interval. Solving this problem is useful in pertinent indoor settings. For example, the indoor semantic locations in question can be the regions in a large exhibition where different items are placed. The top-k regions with highest flows indicate which items are the most popular, and they can be used to make recommendations to future visitors or to optimize the exhibition selections. As another example, the indoor semantic locations can be the individual shops in a large shopping mall. Knowing the most popular semantic locations is useful for the mall management, e.g., to decide the space rental prices.

In the setting of our study, we use a type of indoor mobility data that can be obtained from multiple indoor positioning technologies. To put it briefly, the mobility information of an object at a past time t is captured by a set of samples in the format $(loc, prob)$. Such a sample means that the object is at location loc with a probability $prob$ at time t. Note that loc here is a *point location*, whereas the indoor semantic location mentioned above is a *region location*. We further differentiate them in Section 2.1.

The aforementioned data format is often seen in the indoor positioning services based on wireless infrastructure. In Wi-Fi fingerprinting [25], such samples can be obtained through the weighted k-nearest neighbor method (WkNN) [14] that returns k reference points, where the object is most likely be, with respective probabilities. In wireless packet sniffing [11], [12], the location information gathered by multiple access points (APs) can be represented as samples with probabilities. The purpose of using such a data format is to help achieve better overall positioning effectiveness. In most practical indoor positioning services, the precision of measuring wireless signals is limited by the deployed APs that are mainly designed for network access usage [12], [25]. Also, wireless signals are highly sensitive to environmental factors such as temperature, humidity, AP deployment [6], [15]. In an indoor space where such factors are dynamic, wireless signals fluctuate a lot rather than being stable. To address the problems, using probabilistic samples in a location report is a natural option that can make the location report more reliable [14], [37].

In general, the problem to study is nontrivial due to two challenges. The first challenge is the difficulty in obtaining *reliable* flow values for the indoor semantic locations. On the one hand, the inherent uncertainty in indoor mobility data renders it impossible to directly count the number of objects in an indoor region. The data available at a time t is uncertain as there can be multiple positioning samples. Also, for the time intervals where there is no

- *H. Li, L. Shou, G. Chen, and K. Chen are with the College of Computer Science, Zhejiang University, China. E-mail: {lihuancs, should, cg, chenk}@zju.edu.cn*
- *H. Lu is with the Department of Computer Science, Aalborg University, Denmark. He is the corresponding author. E-mail: luhua@cs.aau.dk*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2875096, IEEE Transactions on Knowledge and Data Engineering

mobility data available for an object, we have no straightforward information about the object's whereabouts. On the other hand, indoor spaces are characterized by walls, doors, rooms, and other entities, which altogether create the unique indoor topology that constrains as well as allows indoor movements. Therefore, unlike outdoor Euclidean spaces and spatial networks, indoor spaces are modeled differently [17], [23], [26], [34], [35] and computing indoor flows must consider the indoor topology appropriately. Note that the aforementioned data uncertainty takes effect in the context of indoor topology, which renders the flow computing even more complex. Hence, the data uncertainty together with indoor topology entails an appropriate formulation of indoor flows.

The second challenge comes from the heavy workloads in the flow computing. In our setting, the volume of indoor mobility data is very large due to the sampling nature of positioning data and the large numbers of indoor moving objects. It is thus computationally expensive to consider all indoor moving objects and their mobility data when we compute the flow for a particular semantic location in a given indoor space. Effective data pruning is certainly needed such that the most popular semantic locations can be found efficiently by computations that only involve the relevant moving objects.

We propose a number of novel techniques to address these challenges. First, we formulate the definition of indoor flows by taking into account both data uncertainty and indoor topology. Given a time interval $[t_s, t_e]$, we identify all possible indoor paths for an object from its positioning samples in $[t_s, t_e]$. Accordingly, the indoor flow for an indoor semantic location is calculated in terms of the number of such paths that go through that location. Second, we design data structures that bridge the gap between raw indoor positioning data and indoor topology in our setting. The structures make it easy to access the data needed in the flow computing. Third, we design a data reduction method that can significantly reduce the number of possible indoor paths to consider and thus is able to improve the efficiency of flow computing by orders of magnitude. Fourth, by using the data structures and data reduction method, we propose the search algorithms to find the top-k popular indoor semantic locations, i.e., those with the highest flow values. Our best search algorithm is able to aggressively prune moving objects that are irrelevant to the flows of the top-k popular semantic locations.

We conduct extensive experiments to evaluate our proposals using both real and synthetic data. The results show that our overall solution is efficient and scalable. Also, our approach to indoor flows is effective in that the top-k results returned by our approach are highly consistent with ground truth.

We make the following contributions in this paper.

- We formulate the indoor flow definition and the indoor top-k popular location query that returns the indoor semantic locations with the highest flow values in a past time interval (Section 2).
- We design a complete set of techniques for efficiently computing the flows for individual indoor semantic locations, including the data structures that facilitate accessing the relevant data, a data reduction method that reduces the scale of intermediate data to process, and an overall flow computing algorithm (Section 3).
- We design search algorithms for answering the indoor top-k popular location query (Section 4).
- We conduct extensive experiments to evaluate our proposals. The results verify the efficiency, scalability, and effectiveness of our approach (Section 5).

In addition, Section 6 reviews the related work; Section 7 concludes the paper and discusses future work.

2 Preliminaries

Table 1 lists the notations used throughout this paper.

Table 1
Notations

Symbol	Meaning
$o \in O$	an indoor moving object
p, $loc \in \mathcal{L}_P$	an indoor P-location
$c \in C$	an indoor cell
$s \in \mathcal{L}_S$	an indoor S-location
$q \in Q$	an indoor S-location in a query set
X	a sample set in a positioning record
$\pi_l(X)$	a P-location set of a sample set X
$\mathcal{X} = (X_1, \ldots, X_n)$	a positioning sequence of sample sets
$\phi = (loc_1, \ldots, loc_n)$	a (possible) path of an indoor object
$\phi \rightsquigarrow q$	a path ϕ passes an S-location q

2.1 Indoor Space Locations

An indoor space is naturally divided into *indoor partitions* like rooms, hallways or staircases by indoor entities like walls and doors. For the sake of simplicity, we treat hallways or staircases as rooms and use room and partition interchangeably in this paper.

We differentiate two kinds of indoor locations. *Semantic locations* (S-locations) refer to those region locations that are defined by users and interesting to applications. Such region locations are usually associated with some practical semantics, e.g., a lobby in an office building, the first-aid site in a large shopping mall, etc. *Positioning locations* (P-locations) refer to those point locations returned by an indoor positioning system. Unlike outdoor GPS, the P-locations returned by an indoor positioning system are discrete and often determined from a set of pre-defined positions [24]. This localization discreteness is generally due to the limited labor that is unable to survey an indoor space entirely [14], [25]. For example, in the calibration phase of a fingerprinting based positioning system, the signal feature data for training is only collected in some reference points pre-selected in an indoor space. In the subsequent positioning phase, the reference points whose collected signal data matches the current state best are returned as the current possible locations. We use $\mathcal{L}_P$ and $\mathcal{L}_S$ to denote the sets of P-locations and S-locations, respectively.

We further distinguish P-locations into two subclasses according to their properties with respect to indoor topology. A set of **partitioning P-locations** altogether partition the indoor space into *cells* [1] in that an object cannot move from one cell to another without passing one of these P-locations. In contrast, **presence P-locations** do not partition the indoor space but simply imply the presence of a positioned object.

Example 1. Referring to the example in Figure 1, the floor plan is divided into six indoor partitions: rooms r_1 to r_5 and hallway r_6. Each partition may be a region of interest and can be regarded as an S-location. Two partitioning P-locations 4 and 9 (denoted as p_4 and p_9) located at room doors are used as reference points in Wi-Fi based positioning. They result in a cell c_1 (shaded) that consists of partitions r_1 and r_2 such that an object cannot enter or leave cell c_1 without being positioned at either of this two P-locations. In contrast, P-locations 6 and

1. A cell defined in this paper is an indoor partition or a combination of adjacent indoor partitions.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2875096, IEEE Transactions on Knowledge and Data Engineering

8 (p_6 and p_8) in partition r_6 and P-location 7 (p_7) in cell c_1 are presence P-locations.

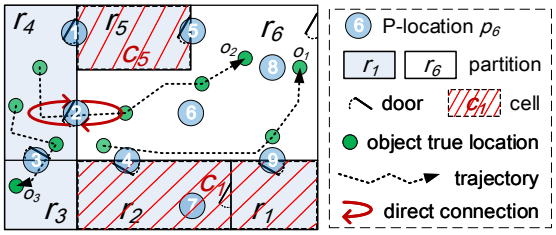


Figure 1. An Example of Indoor Space and Locations

2.2 Indoor Positioning Data

Following the literature [11], [14], [36], [37], the indoor positioning technology in our setting reports the record (oid, X, t) non-periodically, where oid identifies an object, X is a positioning sample set and t is a timestamp. Such a triplet means that the object's location is described by the sample set X at time t. Furthermore, each entry e in the sample set X is in the format of $(loc, prob)$ that means the object is at a P-location loc with probability $prob$. For an arbitrary object and an arbitrary reporting time, $\sum_{e \in X} e.prob = 1$ always holds for the corresponding sample set X. This sample-based positioning approach is very often employed due to its robustness to the dynamic changes in the indoor environment [6], [14], [25]. Given a sample set X, we use $\pi_l(X)$ to denote its P-location set $\{e.loc \mid e \in X\}$.

For a given indoor space of interest, the historical positioning records of indoor moving objects are stored in a table called *Indoor Uncertain Positioning Table* (IUPT). A possible IUPT corresponding to the example in Figure 1 is shown in Table 2. Each object's whereabouts is modeled by a set of probable locations at different reporting time. For example, o_2's P-location set $\pi_l(X)$ is $\{p_1, p_2\}$ at time t_1 and changes to $\{p_2, p_4\}$ at time t_3.

Table 2
Indoor Uncertain Positioning Table

oid , X , t	oid , X , t
o_1 , $\{(p_4, 1.0)\}$, t_1	o_1 , $\{(p_8, 1.0)\}$, t_4
o_2 , $\{(p_1, 0.5), (p_2, 0.5)\}$, t_1	o_2 , $\{(p_5, 0.3), (p_6, 0.6), (p_8, 0.1)\}$, t_5
o_3 , $\{(p_2, 0.6), (p_3, 0.4)\}$, t_2	o_3 , $\{(p_2, 0.4), (p_3, 0.6)\}$, t_5
o_1 , $\{(p_9, 1.0)\}$, t_3	o_2 , $\{(p_5, 0.2), (p_6, 0.3), (p_8, 0.5)\}$, t_6
o_2 , $\{(p_2, 0.7), (p_4, 0.3)\}$, t_3	o_3 , $\{(p_3, 1.0)\}$, t_8

2.3 Problem Formulation

Given a particular indoor S-location q, it is interesting to know how many objects appeared in it during a past time interval $[t_s, t_e]$. As objects are captured in uncertain mobility data, it is not possible to count objects directly or precisely for q. To this end, appropriate alternatives are needed. For simplicity, we assume that t_s and t_e are aligned with sampling time in the indoor positioning.

First, we define the uncertainty-aware *object presence* through the following steps.

1) We obtain the sequence of an object o's sample sets $\mathcal{X} = (X_1, \ldots, X_n)$ during the time interval $[t_s, t_e]$, where X_1 corresponds to t_s and X_n corresponds to t_e ($n \geq 2$).
2) We consider possible paths in the Cartesian product of all relevant P-location sets, i.e., $\pi_l(X_1) \times \ldots \times \pi_l(X_n)$. The total number of such paths can be large in theory, but we can quickly rule out those invalid candidates according to the indoor topology and obtain a set P of *valid* possible paths. The details are to be given in Section 3.3.
3) For each possible path $\phi_i = (loc_1^i, \ldots, loc_n^i) \in P$, we calculate its probability as $pr_i = \prod_{1 \leq j \leq n} prob_j^i$ where $prob_j^i$ is the probability associated with P-location loc_j^i in the corresponding sample set X_j.
4) Given an S-location $q \in \mathcal{L}_S$, we use $pr_{\phi_i \rightsquigarrow q}$ to denote the *pass probability* that a path ϕ_i passes q. Subsequently, for the object o, its **object presence** in q is calculated as

$$\Phi_{t_s,t_e}(q, o) = \frac{\sum_{\phi_i \in P}(pr_{\phi_i \rightsquigarrow q} \cdot pr_i)}{\sum_{\phi_i \in P} pr_i} \quad (1)$$

For an S-location q and a path $\phi = (loc_1, \ldots, loc_n)$, we calculate the pass probability $pr_{\phi \rightsquigarrow q}$ through the following steps.

1) For each sequential P-location pair (loc_j, loc_{j+1}) from ϕ, we find a set C of cells in which each cell covers a *direct* connection from loc_j to loc_{j+1}. The probability that (loc_j, loc_{j+1}) passes q is defined as $pr_{loc_j, loc_{j+1} \rightsquigarrow q} = \frac{|\{c \in C | c \text{ covers } q\}|}{|C|}$. Section 3.1.2 presents the data structure for finding such set C for a given P-location pair (loc_j, loc_{j+1}).
2) Excluding the probability that none of the sequential P-location pairs in path ϕ passes q, ϕ's **pass probability** with respect to q is calculated as

$$pr_{\phi \rightsquigarrow q} = 1 - \prod_{1 \leq j \leq n-1}(1 - pr_{loc_j, loc_{j+1} \rightsquigarrow q}) \quad (2)$$

In Equation 2, the pass probability $pr_{\phi_i \rightsquigarrow q} \leq 1$ always holds. Consequently, for any S-location q and any object o, we have the object presence $\Phi_{t_s,t_e}(q, o) \leq 1$ according to Equation 1.

Example 2. Referring to Table 2, an object o_3 has 4 possible paths during the time interval $[t_1, t_8]$, i.e., $\phi_1 = (p_2, p_2, p_3)$, $\phi_2 = (p_2, p_3, p_3)$, $\phi_3 = (p_3, p_2, p_3)$ and $\phi_4 = (p_3, p_3, p_3)$, with respective probabilities 0.24, 0.36, 0.16 and 0.24. In particular, ϕ_1's probability is $0.6 \times 0.4 \times 1.0 = 0.24$. Moreover, ϕ_1 contains two sequential P-location pairs (p_2, p_2) and (p_2, p_3). Considering (p_2, p_2), two direct connections (the two arrows around p_2 in Figure 1) are found: one is covered by r_6 and the other by r_4. Therefore, we have $pr_{p_2,p_2 \rightsquigarrow r_6} = pr_{p_2,p_2 \rightsquigarrow r_4} = 1/2$. Likewise, for pair (p_2, p_3), $pr_{p_2,p_3 \rightsquigarrow r_4} = 1$ and $pr_{p_2,p_3 \rightsquigarrow r_6} = 0$. According to Equation 2, $pr_{\phi_1 \rightsquigarrow r_6} = 1 - (1 - 1/2) \cdot (1 - 0) = 0.5$. Likewise, we have $pr_{\phi_2 \rightsquigarrow r_6} = pr_{\phi_3 \rightsquigarrow r_6} = pr_{\phi_4 \rightsquigarrow r_6} = 0$. According to Equation 1, the presence $\Phi_{t_1,t_8}(r_6, o_3) = 0.5 \cdot 0.24 = 0.12$. Given another S-location r_1, we have $\Phi_{t_1,t_8}(r_1, o_3) = 0$ as none of o_3's possible paths has chance to pass the S-location r_1.

Based on the concept of object presence, we estimate the expected number of objects that have been in an S-location and define *indoor flow* accordingly.

Definition 1 (Indoor Flow). Given an S-location $q \in \mathcal{L}_S$, a set O of indoor moving objects, and a time interval $[t_s, t_e]$, the indoor flow for q is $\Theta_{t_s,t_e,O}(q) = \sum_{o \in O} \Phi_{t_s,t_e}(q, o)$.

Example 3. From Table 2, we find 3 objects $o_1, o_2, o_3 \in O$. Following the example to process o_3 in Example 2, we calculate o_1's and o_2's presence respectively. For o_1, we find only one valid path (p_4, p_9, p_8), and have $\Phi_{t_1,t_8}(r_1, o_1) = 0.5$ and $\Phi_{t_1,t_8}(r_6, o_1) = 1$ according to Equations 1 and 2. Likewise, for o_2, we have $\Phi_{t_1,t_8}(r_1, o_2) = 0$ and $\Phi_{t_1,t_8}(r_6, o_2) = 0.85$ by calculating on all its valid paths. As a result, S-location r_6's indoor flow is $\Theta_{t_1,t_8,O}(r_6) = \sum_{1 \leq i \leq 3} \Phi_{t_1,t_8}(r_6, o_i) = 1 + 0.85 + 0.12 = 1.97$, and S-location r_1's indoor flow is $\Theta_{t_1,t_8,O}(r_1) = \sum_{1 \leq i \leq 3} \Phi_{t_1,t_8}(r_1, o_i) = 0.5 + 0 + 0 = 0.5$.

Our research problem is defined as follows.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2875096, IEEE Transactions on Knowledge and Data Engineering

Problem 1 (Top-k Popular Location Query, TkPLQ). Given a set Q of indoor semantic locations, an indoor uncertain positioning table IUPT for a set O of indoor moving objects, and a time interval $[t_s, t_e]$, an indoor top-k popular location query returns k S-locations in a k-subset Q_k such that $\forall q \in Q_k, \forall q' \in Q \setminus Q_k, \Theta_{t_s,t_e,O}(q) \geq \Theta_{t_s,t_e,O}(q')$.

Example 4. Referring to the example in Figure 1, given a query set $Q = \{r_1, r_6\}$, as $\Theta_{t_1,t_8,O}(r_1) = 0.5 < \Theta_{t_1,t_8,O}(r_6) = 1.97$, a top-1 query during the time interval $[t_1, t_8]$ returns a room r_6 as the most popular indoor semantic location.

3 Computing Indoor Flows For Individual S-Locations

This section presents the techniques for efficiently computing indoor flows for individual S-locations. Section 3.1 details the data structures that facilitate the relevant computations, Section 3.2 gives a method to reduce the scale of data to process, and Section 3.3 elaborates on the overall flow computing algorithm.

3.1 Data Structures

Our problem definition (Section 2.3) involves possible paths that consist of P-locations, whereas the query set is defined on the S-locations. To enable the estimation of the probability that a path passes an S-location, we should properly capture the relationship between those two types of indoor locations in the context of indoor topology. To this end, we first devise an *indoor space location graph* in Section 3.1.1 that organizes the P-locations, S-locations, and cells into an indoor space topological model. Subsequently, we design an *indoor location matrix* in Section 3.1.2 in order to facilitate searching the relevant cells and S-locations of two sequential P-locations in a path.

3.1.1 Indoor Space Location Graph

The indoor space location graph G_{ISL} generalizes the concept of RFID deployment graph [17] to accommodate the indoor locations discussed in this paper. Representing the topological connectivity of the indoor space at the level of indoor cells, G_{ISL} is defined as a labeled graph (C, E, ℓ_e) where:

1) C is the set of the vertices. Each vertex corresponds to an indoor cell c that results from the partitioning P-location(s) as described in Section 2.1.
2) E is the set of edges, i.e., $E = \{\langle c_i, c_j\rangle \mid c_i, c_j \in C\}$.
3) $\ell_e : E \rightarrow 2^{\mathcal{L}_P}$ maps an edge to a set of P-locations. In particular, if $c_i \neq c_j$, edge $\langle c_i, c_j\rangle \in E$ indicates that cells c_i and c_j are connected in that an object can move from c_i to c_j (or the opposite way) without getting into a third cell. In this case, $\ell_e(\langle c_i, c_j\rangle)$ gives the set of partitioning P-locations that lead to the dividing of cells c_i and c_j. For a loop edge $\langle c_i, c_i\rangle \in E$, $\ell_e(\langle c_i, c_i\rangle)$ gives the set of presence P-locations that are fully covered by the cell c_i.

Figure 2 shows the indoor space location graph for the example floor plan in Figure 1. We use two mappings to facilitate the search between the cells and the S-locations. $C2S : C \rightarrow 2^{\mathcal{L}_S}$ maps a cell to the set of S-locations it contains, whereas $Cell : \mathcal{L}_S \rightarrow C$ maps an S-location to the *parent cell* that contains it. Given an object's possible path ϕ, we consider that it passes an S-location q if we know it passes q's parent cell $Cell(q)$. This way simplifies the computation of the probability that an object passes an S-location q. Here, we assume that an S-location corresponds to one parent cell. Nevertheless, our mappings and relevant techniques can be extended to support an S-location involving multiple cells.

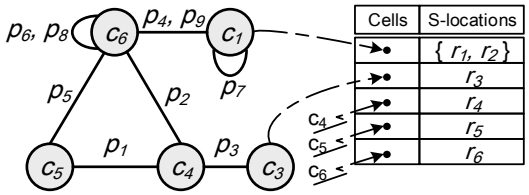


Figure 2. Indoor Space Location Graph

Our data structure gives access to both S-locations and cells. S-locations support application-level specifications while cells facilitate generic indoor flow computing. An advantage of the design is the separation of the specific user needs and the fundamental flow computations on indoor mobility data. Users are allowed to define a set of S-locations for a new task by only reconstructing the corresponding mappings without getting to the underlying flow computing approach.

The indoor space location graph can be defined as a directed graph in order to support door directionality. Here we omit such details and use the undirected graph only. The techniques proposed in this paper can easily be adapted to handle door directionality.

3.1.2 Indoor Location Matrix

In order to determine if an S-location (or its parent cell) is passed by a given P-location sequence (i.e., a path), we define the indoor location matrix M_{IL} as follows. It is a N-by-N upper triangular matrix [2], where $N = |\mathcal{L}_P|$ is the total number of P-locations. Given two different P-locations $p_i, p_j \in \mathcal{L}_P$, the following properties hold:

1) $M_{IL}[p_i, p_i]$ gives the adjacent cells to p_i if p_i is a partitioning P-location. Otherwise, $M_{IL}[p_i, p_i]$ gives the cell that contains p_i.
2) $M_{IL}[p_i, p_j]$ gives the cells through which one can reach p_j from p_i without involving any other cells or P-locations.
3) $M_{IL}[p_i, p_j] = \varnothing$ if p_i and p_j are not connected by a common cell. In other words, one can reach p_i from p_j only by going through more than one cell.

Referring to the example in Figure 1, the corresponding indoor location matrix is shown in Figure 3. Here, $M_{IL}[p_4, p_9] = \{c_1, c_6\}$ as cell c_1 connects P-locations p_4 and p_9, and so does cell c_6. One can reach p_9 from p_4 (or in the opposite way) without leaving cell c_1 or c_6. In other words, if we see two positioning samples involving p_4 and p_9 sequentially, we can tell that the object is in either cell c_1 or c_6. For presence P-location p_8, we have $M_{IL}[p_8, p_8] = c_6$ as p_8 is inside cell c_6. On the other hand, $M_{IL}[p_3, p_4] = \varnothing$ indicates that there is no immediate path from p_3 to p_4. Indeed, one has to go through cells c_4 and c_6 to reach p_4 from p_3. The indoor location matrix can be easily built by utilizing the connectivity information captured in G_{ISL}. We omit such details due to the space limit.

We further discuss how to downsize M_{IL} as its dimensionality is as large as $|\mathcal{L}_P|$. Recall that in Section 3.1.1, the mapping $\ell_e(\langle c_i, c_j\rangle)$ maps an edge of graph G_{ISL} to a set of P-locations, let the set be $locs$. Indeed, $\forall p_k \in \mathcal{L}_P$ and $\forall p_i, p_j \in locs$, $M_{IL}[p_k, p_i] = M_{IL}[p_k, p_j]$ always holds, i.e., p_i and p_j are logically equivalent in constructing M_{IL}. We say p_i and p_j are *equivalent P-locations*, denoted by $p_i \equiv p_j$. Referring to the example in Figure 1, for a given P-location p_4, we find $p_6 \equiv p_8$

2. It is a non-triangular matrix if door directionality is considered.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2875096, IEEE Transactions on Knowledge and Data Engineering

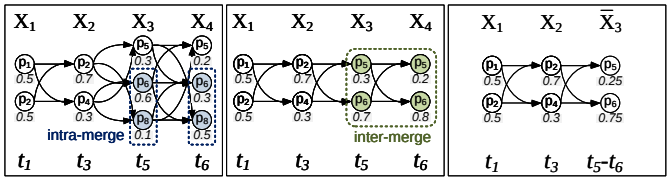
	$\mathbf{p}_1$	$\mathbf{p}_2$	$\mathbf{p}_3$	$\mathbf{p}_4$	$\mathbf{p}_5$	$\mathbf{p}_6$	$\mathbf{p}_7$	$\mathbf{p}_8$	$\mathbf{p}_9$
$\mathbf{p}_1$	$\{c_4,c_5\}$	c_4	c_4	$\varnothing$	c_5	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$
$\mathbf{p}_2$		$\{c_4,c_6\}$	c_4	c_6	$\varnothing$	c_6	$\varnothing$	c_6	c_6
$\mathbf{p}_3$			$\{c_3,c_4\}$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$
$\mathbf{p}_4$				$\{c_1,c_6\}$	c_6	c_6	c_1	c_6	$\{c_1,c_6\}$
$\mathbf{p}_5$					$\{c_5,c_6\}$	c_6	$\varnothing$	c_6	c_6
$\mathbf{p}_6$						c_6	$\varnothing$	c_6	c_6
$\mathbf{p}_7$							c_1	$\varnothing$	c_1
$\mathbf{p}_8$								c_6	c_6
$\mathbf{p}_9$									$\{c_1,c_6\}$

Figure 3. Indoor Location Matrix

in searching c_6 and $p_4 \equiv p_9$ in searching $\{c_1, c_6\}$. Consequently, we can merge all such equivalent P-locations in an edge of G_{ISL} to eliminate their redundancy on searching the identical set of cells. As a result, M_{IL} is reduced to an M-by-M matrix, where $M = |G_{ISL}.E|$ is the number of G_{ISL}'s edges [3]. Note that $M \ll |\mathcal{L}_P|$ as the number of P-locations is usually much greater than that of vertices (or edges) in the corresponding G_{ISL}. By this merging, we can downsize M_{IL} and consequently reduce the scale of possible paths to be generated. More details for the merging are to be given in Section 3.2.

3.2 Data Reduction Method

Recall the object presence defined in Section 2.3, the set of possible paths formed by Cartesian product can have an explosive increasing size with the length of query time interval. Large path sets can lead to slow computation of indoor flows and thus become the performance bottleneck. Given a positioning sequence $\mathcal{X} = (X_1, \ldots, X_n)$, the *maximum* number of generated paths is as large as $\prod_{1 \le i \le n} |\pi_l(X_i)|$. An example of constructing object o_2's paths is shown in Figure 4(a), 4 sample sets within $[t_1, t_8]$ are searched from IUPT, incurring up to 32 generated paths to process. To reduce the number of possible paths to be involved in the flow computing, we propose a data reduction method.



(a) raw sequence |P| = 32 (b) after intra-merge |P| = 16 (c) after inter-merge |P| = 8

Figure 4. Example of Data Reduction on Object o_2's Positioning Data

We bring up two operations to reduce the number of paths generated. On the one hand, the path construction can benefit from a smaller P-location set $\pi_l(X_i)$ at each X_i. This is done by merging the samples of those equivalent P-locations (discussed in Section 3.1.2) inside each sample set. We call it *intra-merge*.

On the other hand, it is useful to reduce the length of the sequence $\mathcal{X}$. People may stay in the same place for a long time and their positions only change slightly, resulting in a sequence of sample sets that contain the identical set of P-locations. In this case, we should merge such consecutive sample sets into one because they point to almost the same underlying whereabouts within a time period [4]. We call this operation *inter-merge*. Particularly, for each common P-location, we approximately estimate its substituted probability in the new merged sample set as the mean of its probabilities in all original sample sets. Our approximate probability estimation simplifies the information held by the original sample sets we merged; However, it can avoid generating a huge number of repetitious paths that result from the repeated positioning reports in the same place.

Figure 4 illustrates the intra-merge and inter-merge operations. Referring to Figure 4(a), P-locations p_6 and p_8 included in $\pi_l(X_3)$ are equivalent in searching the relevant cells from M_{IL}, thus p_8's probability is merged with p_6's [5]. The same operation is applied to X_4. The probability after an intra-merge is the sum of all original probabilities involved in the merge. Subsequently, the consecutive X_3 (at time t_5) and X_4 (at time t_6) in Figure 4(b) can be merged since they contain the identical P-location set $\{p_5, p_6\}$. By computing each common P-location's mean of probabilities in different sets, we obtain a new merged sample set $\overline{X}_3$ where the substituted probabilities are $prob_{p_5} = (0.3 + 0.2)/2 = 0.25$ and $prob_{p_6} = (0.7 + 0.8)/2 = 0.75$. Finally, the maximum size of generated paths in Figure 4(c) decreases to 8 from 32.

In addition to the merge operations, we also reduce the whole positioning sequence for the objects whose location reports are irrelevant to all S-locations in the query set. Taking object o_3 in Table 2 as an example, its P-location sets are $\{p_2, p_3\}$, $\{p_2, p_3\}$ and $\{p_3\}$ during the interval $[t_1, t_8]$. By checking M_{IL} in Figure 3, we get three cells c_3, c_4 and c_6 that involve p_2 and/or p_3. Furthermore, from the mapping in Figure 2, we get the *possible semantic locations* (PSLs) that o_3 may have passed are r_3, r_4 and r_6. If a query location set contains none of its PSLs, object o_3 can be safely excluded from flow computing. Therefore, using PSLs can quickly prune those irrelevant objects (and indirectly all its paths) that cannot pass the semantic locations in the query set. To find all PSLs for an object, we can perform a quick scan over its sample set sequence without constructing any path. This procedure can be easily integrated with the merge operations.

The data reduction method in Algorithm 1 obtains a sequence $\mathcal{X}$ of sample sets (line 1) and returns the reduced sequence $\mathcal{X}'$ with its PSLs $psls$ (initialized in line 2). Also, a temporary sequence $\mathcal{X}_{merge}$ is used to hold those consecutive sample sets for the inter-merge (initialized in line 3). The algorithm iterates through each sample set X_i and first calls `IntraMerge` for it (lines 4–5). After each intra-merge, all PSLs found for X_i are added to $psls$ (lines 6–7). Next, the previous sample set X_{tail} is obtained from $\mathcal{X}_{merge}$ (line 8) and compared to X_i. If their P-location sets are not identical (line 9), i.e., $\mathcal{X}_{merge}$ can no longer include any subsequent sample set for a merge, function `InterMerge` is called for $\mathcal{X}_{merge}$ and the resultant sample set is added to $\mathcal{X}'$ (line 10). The sequence $\mathcal{X}_{merge}$ is emptied as long as the inter-merge is done (line 10). No matter whether an inter-merge for the current $\mathcal{X}_{merge}$ is conduct or not, X_i is added to $\mathcal{X}_{merge}$ for further determination (line 11). An inter-merge should also be performed at the end of the whole iteration (line 12) since there are no more succeeding sample sets.

The `IntraMerge` procedure handles the samples inside each sample set X (lines 14–21). If a subset $\widetilde{X}$ contains equivalent P-locations that refer to identical cells in M_{IL} (line 16), the subset $\widetilde{X}$ is removed from X (line 17), a P-location with the smallest subscript from $\pi_l(\widetilde{X})$ is chosen as the representative $\widetilde{loc}$ (line 18), and the sample probabilities in $\widetilde{X}$ are summed up to $\widetilde{pr}$ (line 19). Afterwards, a new sample $\widetilde{e}(\widetilde{loc}, \widetilde{pr})$ is added back to X (line 20).

The `InterMerge` procedure works as follows (lines 22–30). If the sequence $\mathcal{X}_{merge}$ to be merged has only one sample set, it just returns that sample set (lines 23–24). Otherwise, for each

3. For readability, we still use the original M_{IL} to explain our algorithms.

4. The indoor flow defined in this study is independent of the dwell time of an object at a particular S-location.

5. To maintain the consistency in search, we keep the P-location with a smaller subscript after a merge.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2875096, IEEE Transactions on Knowledge and Data Engineering

Algorithm 1 ReduceData(Sample set sequence $\mathcal{X}$, Indoor semantic locations Q)

1: $(X_1, \ldots, X_n) \leftarrow \mathcal{X}$
2: initialize a sample set sequence $\mathcal{X}'$; initialize a set $psls \leftarrow \varnothing$
3: initialize a sample set sequence $\mathcal{X}_{merge}$
4: **for** i from 1 to n **do**
5: $X_i \leftarrow$ IntraMerge(X_i)
6: $psls' \leftarrow \bigcup_{loc \in \pi_l(X_i)} C2S(M_{IL}[loc, *] \cup M_{IL}[*, loc])$
7: $psls \leftarrow psls \cup psls'$
8: $X_{tail} \leftarrow \mathcal{X}_{merge}$.tail()
9: **if** $X_{tail} \neq null$ **and** $\pi_l(X_i) \neq \pi_l(X_{tail})$ **then**
10: add InterMerge($\mathcal{X}_{merge}$) to $\mathcal{X}'$; $\mathcal{X}_{merge}$.empty()
11: add X_i to $\mathcal{X}_{merge}$
12: **if** $i = n$ **then** add InterMerge($\mathcal{X}_{merge}$) to $\mathcal{X}'$
13: **if** $psls \cap Q \neq \varnothing$ **then return** $\langle \mathcal{X}', psls \rangle$ **else return** $\langle null, null \rangle$
14: **function** IntraMerge(Sample set X)
15: **for** each subset $\widetilde{X} \subseteq X$ **do**
16: **if** $|\widetilde{X}| \geq 2$ and $\pi_l(\widetilde{X})$ are equivalent P-locations **then**
17: $X \leftarrow X \setminus \widetilde{X}$
18: $\widetilde{loc} \leftarrow loc$ with smallest subscript in $\pi_l(\widetilde{X})$
19: $\widetilde{pr} \leftarrow \sum_{e \in \widetilde{X}} e.prob$
20: add the merged sample $\widetilde{e}(\widetilde{loc}, \widetilde{pr})$ to X
21: **return** X
22: **function** InterMerge(Sample set sequence $\mathcal{X}_{merge}$)
23: **if** $|\mathcal{X}_{merge}| = 1$ **then** ▷ One sample set, return directly
24: **return** $\mathcal{X}_{merge}$.front()
25: **else** ▷ To merge the consecutive sample sets
26: initialize a new sample set $X \leftarrow \varnothing$
27: **for** each P-location loc in $\pi_l(\mathcal{X}_{merge}$.front()) **do**
28: $pr \leftarrow \frac{\sum_{X' \in \mathcal{X}_{merge}} X'[loc].prob}{|\mathcal{X}_{merge}|}$
29: add the formed sample $e(loc, pr)$ to X
30: **return** X

common P-location loc (line 27), it computes the mean of probabilities pr from all corresponding samples $X'[loc], X' \in \mathcal{X}_{merge}$ (line 28), and adds the formed sample $e(loc, pr)$ to the new set X (line 29). When all locs have been processed, X is returned as the merged sample set (line 30).

At the end, if none of the S-locations in $psls$ is included in the query set, $null$ is returned to indicate that this sequence will not be used in the flow computing. Otherwise, $psls$ is returned together with $\mathcal{X}'$ (line 13). By calling Algorithm 1, we can significantly reduce the number of objects (sample set sequences) and their generated paths to be processed further in flow computing.

3.3 Flow Computing Algorithm

We are now able to compute indoor flow values for individual S-locations. We use a one-dimensional R-tree (termed the *1DR-tree*) [28] to index the IUPT on its time attribute. Given an S-location q and a time interval $[t_s, t_e]$, Algorithm 2 fetches the positioning records within $[t_s, t_e]$ (line 1), inserts them into an object hash table H_O (lines 2–4), and obtains q's flow by going through all object positioning records temporally relevant (lines 5–21). In particular, for each object, the algorithm calls ReduceData (see Algorithm 1) to obtain its reduced positioning sequence (line 7). Objects whose $psls$ does not overlap with q are excluded from subsequent processing (line 8). The reduced sequence is used to form the path set P described in Section 2.3 (lines 9–15). Specifically, the indoor location matrix M_{IL} is checked to determine if the current path to be generated is valid or not (line 14), and only the valid ones are added to P (line 15) to involve with the succeeding path generation. This way, we can avoid generating many branches of invalid candidates. Afterwards, the algorithm computes the object's presence in the way described in Section 2.3 and adds it to q's overall flow (lines 16–21).

Algorithm 2 Flow(Indoor semantic location q, 1DR-tree $tree$, Query time interval $[t_s, t_e]$)

1: LeafEntrySet $les \leftarrow tree$.RangeQuery($[t_s, t_e]$)
2: initialize a hash table $H_O : \{oid\} \rightarrow \{\mathcal{X}\}$
3: **for** each leaf entry $le \in les$ **do**
4: append $le.X$ to $H_O[le.oid]$
5: $flow \leftarrow 0$
6: **for** each key $oid \in H_O.keys$ **do**
7: $\langle (X_1, \ldots, X_n), psls \rangle \leftarrow$ ReduceData($H_O[oid]$, $\{q\}$)
8: **if** $psls$ is $null$ **then continue**
9: path set $P \leftarrow \{\langle (loc, prob) \rangle \mid (loc, prob) \in X_1\}$
10: **for** i from 1 to n **do**
11: **for** each path $\phi \in P$ **do**
12: remove ϕ from P
13: **for** each sample $e \in X_i$ **do**
14: **if** $M_{IL}[\phi.tail.loc, e.loc] \neq \varnothing$ **then**
15: $\phi' \leftarrow$ append(ϕ, e); add ϕ' to P
16: $pr \leftarrow 0$; $pr_{sum} \leftarrow 0$
17: **for** each path $\phi \in P$ **do**
18: $pr_\phi \leftarrow \Pi_{1 \leq j \leq |\phi|} \phi[j].prob$; $pr_{sum} \leftarrow pr_{sum} + pr_\phi$
19: **if** $pr_{\phi \rightsquigarrow q} > 0$ **then** ▷ ϕ has chance to pass $Cell(q)$
20: $pr \leftarrow pr + (pr_{\phi \rightsquigarrow q} \cdot pr_\phi)$
21: $flow \leftarrow flow + \frac{pr}{pr_{sum}}$
22: **return** $flow$

4 Algorithms for TkPLQ

On the top of the data structures and techniques for computing object presences and indoor flows in Section 3, this section presents the algorithms for processing the TkPLQ. As illustrated in Figure 5, a naive algorithm computes the indoor flow of each

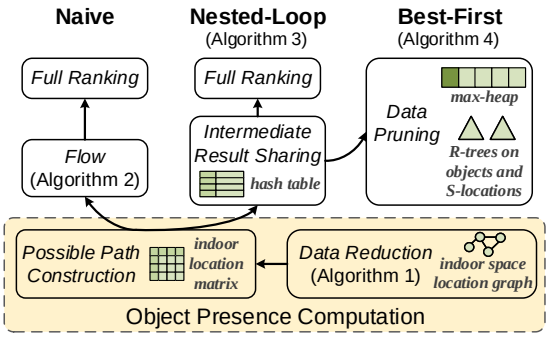


Figure 5. Naive, Nested-Loop, and Best-First Algorithms for TkPLQ

query S-location in Q by calling Algorithm 2, and returns the query S-locations with the top-k highest flow values. This is inefficient as the blind call of Algorithm 2 may process the positioning samples and the relevant paths of the same object repeatedly. Suppose that an object o has gone through two query S-locations $q_i, q_j \in Q$ during the query time interval. In the two calls of Algorithm 2 for q_i and q_j, o's samples and paths are processed twice. To avoid such re-computations, in Section 4.1, we present a nested-loop algorithm that improves the efficiency by sharing the intermediate results. In Section 4.2, we further introduce a best-first algorithm that prunes unpromising query locations and irrelevant moving objects rather than using a full ranking. Figure 5 depicts the relations and differences among the three algorithms.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2875096, IEEE Transactions on Knowledge and Data Engineering

4.1 Nested-Loop Algorithm

Algorithm 3 Nested-Loop(Indoor semantic locations Q, 1DR-tree $tree$, Query time interval $[t_s, t_e]$)

1: LeafEntrySet $les \leftarrow tree$.RangeQuery($[t_s, t_e]$)
2: initialize a hash table $H_O : \{oid\} \rightarrow \{\mathcal{X}\}$
3: **for** each leaf entry $le \in les$ **do**
4: append $le.X$ to $H_O[le.oid]$
5: initialize a hash table $H_Q : Q \rightarrow \{score\}$
6: **for** each key $oid \in H_O.keys$ **do**
7: $\langle (X_1, \ldots, X_n), psls \rangle \leftarrow$ `ReduceData`($H_O[oid]$, Q)
8: **if** $psls$ is $null$ **then continue**
9: path set $P \leftarrow \{\langle (loc, prob) \rangle \mid (loc, prob) \in X_1\}$
10: initialize a hash table $H_\phi : \{path\} \rightarrow 2^Q$
11: **for** i from 1 to n **do**
12: **for** each path $\phi \in P$ **do**
13: remove ϕ from P
14: $list_Q \leftarrow$ remove $H_\phi[\phi]$ from H_ϕ
15: **for** each sample $e \in X_i$ **do**
16: **if** $M_{IL}[\phi.tail.loc, e.loc] \neq \varnothing$ **then**
17: $\phi' \leftarrow$ append(ϕ, e); add ϕ' to P
18: $list'_Q \leftarrow (C2S(M_{IL}[\phi.tail.loc, e.loc]) \cap Q)$
19: $H_\phi[\phi'] \leftarrow list_Q \cup list'_Q$
20: initialize a hash table $H_{ls} : Q \rightarrow \{score\}$
21: $pr_{sum} \leftarrow 0$
22: **for** each path $\phi \in P$ **do**
23: $pr = \Pi_{1 \leq j \leq |\phi|} \phi[j].prob$; $pr_{sum} \leftarrow pr_{sum} + pr$
24: **for** each query S-location $q \in H_\phi[\phi]$ **do**
25: $H_{ls}[q] \leftarrow H_{ls}[q] + (pr_{\phi \rightsquigarrow q} \cdot pr)$
26: **for** each query S-location $q \in H_{ls}.keys$ **do**
27: $H_Q[q] \leftarrow H_Q[q] + \frac{H_{ls}[q]}{pr_{sum}}$
28: **return** the top-k from $H_Q.keys$ with the highest scores

The way to process the TkPLQ in the nested-loop join paradigm is formalized in Algorithm 3. Initially, object positioning records falling in the query time interval are found via a 1DR-tree (line 1), and the positioning sequence of each object is constructed by concatenating all its records (lines 2–4). Subsequently, the algorithm iterates on each object (lines 6–27). In each iteration, the algorithm first calls `ReduceData` to reduce a positioning sequence (line 7) and filters out the irrelevant object with regard to Q (line 8), then forms all valid possible paths upon the reduced sequence (lines 9–19). The valid path generation is ensured by checking M_{IL} (line 16). Meanwhile, the query locations on each path are recorded in a hash table H_ϕ that is local to each encountered object (lines 10 and 18–19). After all valid paths are generated for the current object, the algorithm continues to process each path (lines 20–25), calculating the local temporary scores for each S-location with respect to the current object in process. The local scores are stored in a hash table H_{ls} that maps a query location q to its local score across the current object's all valid paths (lines 20 and 25). Furthermore, according to the definitions in Section 2.3, the local score for the current object is aggregated with the global score obtained from all objects that have been seen so far (lines 26–27). Finally, the algorithm returns the top-k S-locations with highest global scores in H_Q (line 28).

Algorithm 3 returns the top-k results as long as the complete set of objects in H_O have been processed. In fact, the returned top-k query locations usually cover only parts of the whole indoor space, meaning that some objects do not contribute to any of the top-k query locations. Thus, identifying and skipping such objects can accelerate our query processing. We proceed to present a best-first algorithm that gives priority to a set of promising query locations with greater flow estimates, and that can avoid some unnecessary but complex computations only for unpromising query locations.

4.2 Best-First Algorithm

Algorithm 4 Best-First(R-tree R_Q for indoor semantic locations Q, 1DR-tree $tree$, Query time interval $[t_s, t_e]$)

1: LeafEntrySet $les \leftarrow tree$.RangeQuery($[t_s, t_e]$)
2: initialize a hash table $H_O : \{oid\} \rightarrow \{\mathcal{X}\}$
3: **for** each leaf entry $le \in les$ **do**
4: append $le.X$ to $H_O[le.oid]$
5: initialize an in-memory COUNT-aggregate R-tree R_C
6: **for** each key $oid \in H_O.keys$ **do**
7: $\langle \mathcal{X}', psls \rangle \leftarrow$ `ReduceData`($H_O[oid]$, Q)
8: **if** $psls$ is not $null$ **then** ▷ $psls$ overlaps with Q
9: get $psls$'s MBR mbr
10: insert (oid, mbr) to R_C
11: initialize a max-heap $\mathcal{H}$
12: **for** each entry e_Q in $R_Q.root$ **do**
13: $ubFlow \leftarrow 0$; $list \leftarrow \varnothing$
14: **for** each entry e_C in $R_C.root$ **do**
15: **if** $e_Q.mbr$ intersects $e_C.mbr$ **then**
16: $ubFlow \leftarrow ubFlow + e_C.count$
17: add e_C to $list$
18: $\mathcal{H}$.enheap($e_Q, list, ubFlow$)
19: $result \leftarrow \varnothing$
20: **while** $\mathcal{H}$ is not empty **do**
21: $\langle e_Q, list, ubFlow \rangle \leftarrow \mathcal{H}$.deheap()
22: **if** e_Q is a leaf entry **then** ▷ e_Q stores a query S-location
23: **if** $list$ is $null$ **then**
24: add S-location $e_Q.object$ to $result$
25: **if** $|result| = k$ **then return** $result$
26: **else**
27: **if** $list$ contains leaf entries **then**
28: use all objects contained by $list$ to compute $flow$ for the query S-location $e_Q.object$
29: $\mathcal{H}$.enheap($e_Q, null, flow$)
30: **else**
31: `ExpandList`($e_Q, list$)
32: **else**
33: **if** $list$ contains leaf entries **then**
34: **for** each sub-entry $e'_Q \in e_Q.node$ **do**
35: $ubFlow \leftarrow 0$; $list2 \leftarrow \varnothing$
36: **for** each entry $e_C \in list$ **do**
37: **if** $e'_Q.mbr$ intersects $e_C.mbr$ **then**
38: $ubFlow \leftarrow ubFlow + 1$
39: add e_C to $list2$
40: **if** $list2 \neq \varnothing$ **then** $\mathcal{H}$.enheap($e'_Q, list2, ubFlow$)
41: **else**
42: **for** each sub-entry $e'_Q \in e_Q.node$ **do**
43: `ExpandList`($e'_Q, list$)
44: **function** `ExpandList`(Node entry e_Q from R-tree R_Q, Join list $list$) ▷ To expand the join list
45: $ubFlow \leftarrow 0$; $list2 \leftarrow \varnothing$
46: **for** each entry $e_C \in list$ **do**
47: **for** each sub-entry $e' \in e_C.node$ **do**
48: **if** $e_Q.mbr$ intersects $e'.mbr$ **then**
49: $ubFlow \leftarrow ubFlow + e'.count$
50: add e' to $list2$
51: **if** $list2 \neq \varnothing$ **then** $\mathcal{H}$.enheap($e_Q, list2, ubFlow$)

As formalized in Algorithm 4, the best-first algorithm consists of three phases. In the first phase (lines 1–10), the data preparation (lines 1–4) is the same as the counterpart in Algorithm 3. Once

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2875096, IEEE Transactions on Knowledge and Data Engineering

the hash table H_O that holds positioning sequence for each object is constructed, the algorithm iteratively organizes these objects into an in-memory COUNT-aggregate R-tree [32] R_C (lines 5–10). Each non-leaf node entry e in R_C is augmented with a count $e.count$ that stores the number of objects covered in e's child nodes. In particular, for each object (line 6), if its obtained $psls$ (line 7) overlaps with Q (line 8), the MBR that contains $psls$ and the object itself are inserted into R_C (lines 9–10). In the implementation, we use a series of smaller, finer-grained MBRs to represent each $psls$ to speed up the join of trees.

The second phase (lines 11–18) prepares for the join of the query S-location R-tree R_Q and the aggregate R-tree R_C. A max-heap $\mathcal{H}$ is initialized to give higher priority to R_Q entries (groups of query S-locations) that potentially have higher flow values (lines 11 and 18). For each entry e_Q from R_Q, an associated join list (line 13) is built with these R_C entries that intersect with e_Q's MBR (line 17). It is noteworthy that the flow value for any S-location in e_Q can only come from such intersecting R_C entries. When the two tree roots are initially joined (lines 12-17), the $count$s of those R_C entries are used to upper bound the flow estimate $ubFlow$ (line 16), as an object's presence in any S-location can never exceed 1 (see Section 2.3).

The third phase (lines 20–43) carries out the join in an order controlled by the max-heap (lines 20–21). If the current entry e_Q is a leaf entry (lines 22–31), we check its join list. If it is empty, i.e., e_Q's concrete flow value has been computed and the value is higher than those yet to be computed, we add it to the result (line 24). If the result contains k S-locations, the algorithm then terminates (line 25). Otherwise, the join list may either contain leaf entries or non-leaf entries. For the former case (line 27), objects in the join list are loaded in order to compute the concrete flow value of the leaf entry e_Q (line 28). Since each object's samples and paths may overlap different S-locations, the intermediate results of each called object should be shared as presented in Algorithm 3. For the latter case (line 30), `ExpandList` is called to join e_Q with the child entries from the join list. This function (lines 44–51) iterates over the join list and finds out the qualified R_C entries, each of which has its MBR intersecting with e_Q (line 48), and meanwhile, the function uppers bound e_Q's flow estimate with the sum of $count$s from e_Q's intersecting R_C entries (line 49).

If the current entry e_Q is a non-leaf entry (lines 32–43), two cases are differentiated. If the join list contains leaf entries (line 33), each of e_Q's sub-entries gets its flow value overestimated (line 38) when joining with the relevant entries from the join list (lines 34–39). A processed sub-entry e'_Q is only added back to the max-heap if its join list is not empty (line 40). If the join list contains non-leaf entries, `ExpandList` is called for each sub-entry of e_Q (lines 42–43).

5 Experimental Studies

This section reports on our experimental studies. Section 5.1 introduces the settings and performance metrics. Sections 5.2 and 5.3 evaluate our proposals on real and synthetic data, respectively.

5.1 Settings and Performance Metrics

All programs are in Java and run on a computer with a 3.30GHz Core i3 CPU. The possible paths are stored in harddisk as their number can be very large when a long query time interval is used.

We compare our search algorithms, namely Naive (introduced at the beginning of Section 4), NL (Algorithm 3 Nested-Loop) and BF (Algorithm 4 Best-First), with the following alternatives. The *Simple Counting* method (SC) works as follows for each positioning record. It picks the (first) sample with the highest probability and discards all other samples. If the corresponding P-location with the highest probability is contained by an S-location q, q's flow value is incremented by one. The SC-ρ differs from SC only in that it picks all the samples whose probability exceeds a given threshold ρ. Both SC and SC-ρ allow a P-location to be counted in multiple S-locations that all contain it, whereas in SC-ρ more samples and P-locations may be involved in counting. Furthermore, an object may be involved in the same S-location at different times. To be consistent with our indoor flow definition, we count an object only once for each relevant S-location during the entire query time interval. We also design the *Monte Carlo* based method (MC) as follows. It executes a certain rounds of simulations. In each round, it simulates an instance of IUPT in which all positioning records are randomly sampled to be certain (i.e., only a P-location is seen in a record), and computes each query location's flow by constructing valid object paths on the certain records. As a result, the top-k query locations are ranked based on their average flow values in all the simulation rounds.

We investigate both efficiency and effectiveness of the top-k search algorithms mentioned above. For efficiency studies, we run each algorithm for a certain times and compare them in terms of the average running time and *pruning ratio*. We define the pruning ratio as $\sigma = (|O| - |O_f|)/|O|$, where O is the set of all indoor moving objects and O_f contains the objects for which the search algorithm has to compute its presence (see Section 2.3).

We also evaluate the effectiveness of the top-k search with respect to ground truth. We consider two effectiveness metrics: recall and *Kendall coefficient*. Specifically, recall measures the fraction of the ground truth top-k popular semantic locations that are returned in the top-k results. The Kendall coefficient τ captures the similarity between the ranking of the top-k search result (φ_r) and that of the top-k ground truth (φ_g). Let cp be the number of S-location pairs (q_i, q_j) whose rankings in φ_r and φ_g are concordant, i.e., q_i is ranked before (after or in tie with) q_j in both φ_r and φ_g. Let dp be the number of pairs (q_i, q_j) whose rankings in φ_r and φ_g are discordant. The Kendall coefficient is $\tau = (cp - dp)/(0.5k(k-1))$. If the two rankings are identical, τ is 1; if one ranking is the reverse of the other, τ is -1. If φ_r and φ_g do not contain the same set of locations, we extend them to the same set in order to compare them. For example, suppose $k = 3$, φ_r is $\langle A, B, C\rangle$ and φ_g is $\langle B, D, E\rangle$. We extend φ_r to $\langle A, B, C, D, E\rangle$ and φ_g to $\langle B, D, E, A, C\rangle$. The elements we add into either ranking have the same ordering value, e.g., elements A and C are ranked 4th in the modified φ_g. As Naive, NL, BF return the *same* top-k results for the same query, we only run BF when evaluating the search effectiveness.

5.2 Experiments on Real Data

We collected the real dataset from a university building, using the Wi-Fi fingerprinting based positioning algorithm [14] that estimates a mobile client's current location as a small set of pre-selected reference points with respective probabilities. Those reference points have their Wi-Fi signal features most similar to that of the client's current location. Being illustrated as a to n in Figure 6, 14 partitions (9 office rooms and 5 hallways) are selected as S-locations from a $33.9m \times 25.9m$ test floor. We randomly pick 20% (40%, 60%, 80%, or 100%) of all these S-locations to form a query set Q. A total of 75 P-locations (reference points) are used in

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2875096, IEEE Transactions on Knowledge and Data Engineering

the test floor. Considering the floor's topology, 16 of the 75 P-locations are used as partitioning P-locations (blue dots), and the others as presence P-locations (green dots). To facilitate the geometrical computation for determining the topological relationships, we use an in-memory R-tree to store the entities including S-locations, P-locations, and doors. Moreover, our designed indoor space location graph and indoor location matrix are also kept in main memory. Their largest memory consumption is around 147.7 KB.

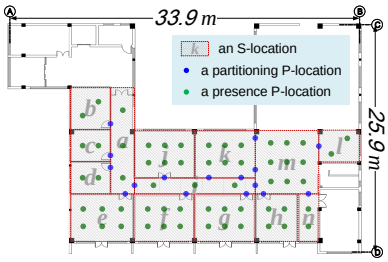


Figure 6. Test Floor of Real Dataset

A total of 35 smartphone users participated in the data collection lasted from April 21st to April 24th, 2015. They were required to specify their actual partitions to obtain the ground truth. We selected a period of 150 minutes from rush hours in a day and got an IUPT with 64,846 positioning records. Each positioning report in IUPT contains up to 4 samples, i.e., the *maximum sample-set size* [14] (mss) is 4. We also use the *maximum positioning period* T to refer to the maximum value of time interval between two consecutive positioning records for a user. According to our survey, T is 3 seconds in our collected data, i.e., the positioning frequency is no less than 1/3 Hz. The average positioning error in the data is about 2.1 meters. For a query time interval, t_s and t_e are randomly decided for a given $\Delta t = t_e - t_s$. The parameter settings are shown in Table 3, where default values are in bold.

Table 3
Parameter Settings on Real Data

Parameters	Settings
k	1, 2, **3**, ..., 8
$\|Q\|$ (% of S-locations)	20%, 40%, **60%**, 80%, 100%
mss	1, 2, 3, **4**
Δt (minute)	**30**, 60, 90

5.2.1 Effect of Using Data Reduction

For our proposed search methods Naive, NL and BF, we implement their corresponding versions that process the original positioning sequence without the data reduction (see Section 3.2). We call them Naive-ORG, NL-ORG and BF-ORG. The performance results for all alternative methods in default parameter setting are reported in Table 4. In SC-ρ, we set $\rho = 0.25$ for the best performance. We control MC's simulation rounds at 900, for which the Kendall coefficient almost increases to a standstill.

Table 4
Performance Comparison in Default Setting

Methods	Running time (sec.)	Pruning ratio (%)	Kendall coefficient τ	Recall (%)
SC	**0.6**	-	0.007	62.2
SC-ρ ($\rho = 0.25$)	1.1	-	0.382	75.6
MC, *900 rounds*	1.7×10^4	-	0.712	86.7
BF	4.4	**59.4**	0.859	93.3
NL	9.5	19.2	same as above.	
Naive	59.1	19.2	same as above.	
BF-ORG	1.4×10^4	50.3	**0.893**	**95.6**
NL-ORG	2.3×10^4	0	same as above.	
Naive-ORG	1.6×10^5	0	same as above.	

It is not surprising that SC and SC-ρ can return the top-k results faster than BF and NL since they do not construct any paths. However, their effectiveness measures are significantly lower. The Kendall coefficient is 0.007 for SC and 0.382 for SC-ρ, which means their rankings of results are highly different from that of the ground truth. They also have a low capability to find the ground truth as the recall is only 62.2% for SC and 75.6% for SC-ρ. In contrast, by applying our uncertainty-aware flow computing, BF and NL's effectiveness measures are significantly higher; BF also achieves a good balance between efficiency and effectiveness.

The results in Table 4 also show the important effect of our data reduction method. Without it, Naive-ORG, NL-ORG and BF-ORG are slower by orders of magnitude than their counterparts with data reduction. Despite its heavy workloads on the original data, BF-ORG prunes 50.3% objects, even much higher than NL. This demonstrates the powerful pruning enabled by the design of our BF algorithm. On the other hand, the data reduction method has very little impact on the search result effectiveness. This is evidenced by the highly similar Kendall coefficient values and recall values for the algorithms with and without data reduction. Compared to our methods with data reduction, MC incurs significantly longer time for its simulations, although it processes on a very small set of generated paths from those certain positioning records in each its simulation round. Besides, both of its effectiveness measures are lower than our proposed methods.

In general, SC and SC-ρ incur little time costs but yield very poor effectiveness. BF-ORG, NL-ORG, Naive-ORG and MC without data reduction all incur extremely long running time.

5.2.2 Effect of Uncertainty in Real Data

Specially, here we discuss the effect of data uncertainty in finding the top-k popular S-locations. We vary the sample capacity of the positioning records as follows. For each a record, if the number of its containing samples exceeds the maximum sample-set size mss, the samples with lower probabilities are removed until only mss samples remain. The reported location becomes certain when mss is 1. In the real data, we do not consider the data uncertainty related to the positioning frequency since it is very high ($\geq$1/3Hz). To further study the effect of data uncertainty, in Section 5.3.1 we vary the maximum positioning period T and the indoor positioning error μ using our synthetic data.

Using defaults for other parameters, we run BF, SC, SC-ρ, and MC with different mss values and report their efficiency performances in Table 5. When we increase mss from 1 to 4, SC and SC-ρ's running time increases steadily as more samples need to be counted; BF's time cost increases more rapidly as the set of possible paths involved becomes larger. Nevertheless, BF can still return the top-3 results within 4.42s. Compared to the others, MC is slower by orders of magnitude. Its running time increases when varying mss from 1 to 2 and stays almost stable with mss up to 4, this is because MC needs more time to randomly select one from the multiple samples in a positioning report.

Table 5
Efficiency Comparison with Different Settings of mss

Methods	Running time (sec.)			
	$mss = 1$	$mss = 2$	$mss = 3$	$mss = 4$
BF	0.18	0.80	2.86	4.42
SC	0.14	0.42	0.53	0.60
SC-ρ ($\rho = 0.25$)	0.17	0.61	0.87	1.12
MC, *900 rounds*	15625	17267	17532	17447

We also investigate the aforementioned methods' effectiveness in different settings of mss. Referring to the results reported in Figure 7, SC's Kendall coefficient τ and recall stay stable when varying mss, as it only uses the sample with the highest

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2875096, IEEE Transactions on Knowledge and Data Engineering

probability in each record and therefore its counting is not affected by the sample capacity. In contrast, SC-ρ, MC and BF's both measures increase significantly when more probabilistic samples are included in the location reports. When mss is 1, the location reports become certain such that some useful information is discarded and underlying positioning errors are magnified. As our real-world location reports were estimated from a set of relatively discrete P-locations (4.48 square meters per P-location), even BF has a τ of 0.462 and recall of 71.1% when each location report contains only one sample. Note that in this case, SC and SC-ρ return the same results, and MC is equivalent to BF without the data reduction. For mss from 2 to 4, both effectiveness measures for SC-ρ, MC and BF increase with more samples introduced, and BF increases more rapidly than the others. The results indicate that our uncertain data model with probabilistic samples is more effective than a certain data model in solving the search problem.

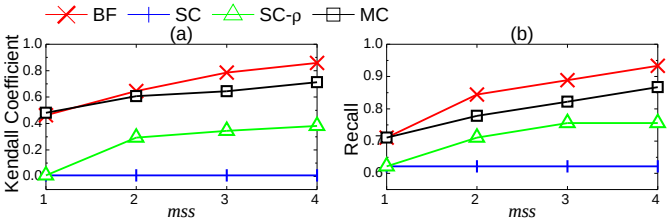


Figure 7. Effectiveness vs. mss on Real Data

5.2.3 Top-k Search Efficiency

We omit the simple counting methods and those inefficient alternatives, and concentrate on NL and BF for efficiency studies on real data. We issue 15 random queries and report their average results.

First, we fix $|Q|$ to 8 (i.e., a fraction 60% of all 14 semantic locations) and Δt to 30 minutes, and vary k. Referring to Figure 8(a), increasing k has little impact on the running time of NL since every object whose $psls$ overlaps with Q is involved in flow computing despite k. BF's time cost increases steadily as we increase k, but it can still return the top-5 results in less than 6.3s. In particular, when k increases to 8, i.e., all locations in Q need to be returned, all the objects must be processed except those that can be filtered out by the data reduction. As a result, BF even costs slightly more time than NL due to its extra operations on the heap and trees. As shown in Figure 8(b), when increasing k, BF's pruning ratio decreases steadily and degrades to that of NL at 19.2% when all S-locations need to be returned. Larger ks require BF to compute flows for more query locations, which tends to involve more objects. The trend is consistent with the running time increase of BF.

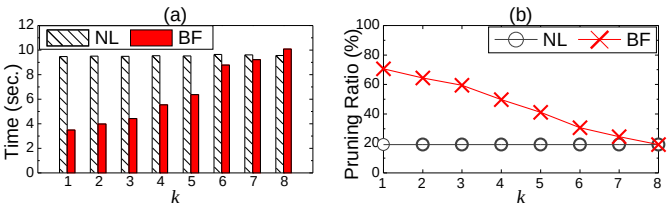


Figure 8. Efficiency vs. k on Real Data

Next, we set k to 3 and Δt to 30 minutes, and vary $|Q|$. Referring to Figure 9(a), both algorithm's running time increases but their difference becomes larger as we increase $|Q|$. Referring to Figure 9(b), both algorithms have to process more objects in computing flows when a larger Q is specified. Also, when all S-locations are included in the query set Q ($|Q| = 100\% \times 14$), all the objects in IUPT need to be processed but BF can terminate early as k is fixed.

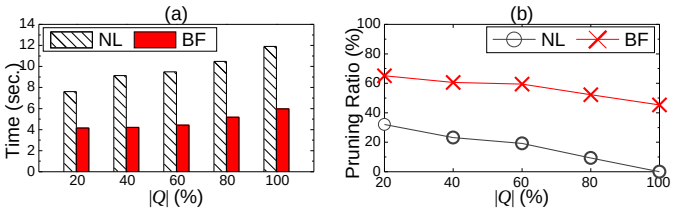


Figure 9. Efficiency vs. $|Q|$ on Real Data

We also set k = 3 and $|Q|$ = 8, and vary Δt. As shown in Figure 10, both algorithms's time cost increase significantly as Δt becomes larger. On one hand, A larger Δt involves more samples to be considered for each object and thus incurs more time to compute the concrete flow values. On the other hand, a larger Δt tends to extend the objects' PSLs, which makes more objects to be involved in the flow computing for a query location. For these two reasons, the time cost grows rapidly with an increasing Δt. Referring to Figure 10(b), the pruning ratio of BF decreases moderately when Δt is varied from 30 minutes to 90 minutes. Since our real data was collected in a relatively small space, a majority of S-locations are included in the query set Q, and therefore the effect of increasing Δt clearly dominates BF's pruning ratio. Nevertheless, when Δt increases to 90 minutes, BF still can prune 16% more objects than NL whose pruning is mostly done by the data reduction. This shows that BF performs well even when the query uses a long time interval.

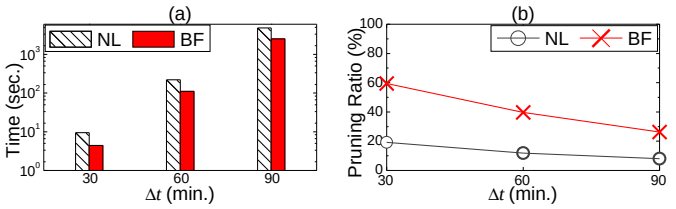


Figure 10. Efficiency vs. Δt on Real Data

5.2.4 Top-k Result Effectiveness

We compare the effectiveness measures of BF, SC, SC-ρ and MC for each setting used in Section 5.2.3.

First, we study the effect of varying k with other parameters fixed to default. Referring to Figure 11(a), BF's Kendall coefficient τ decreases moderately but stays above 0.85 with increasing k up to 3. Overall, τ is still above 0.77. Referring to Figure 11(b), BF's recall is higher than 0.88 for most tested cases. These results verify the high effectiveness of our search on real dataset. Compared

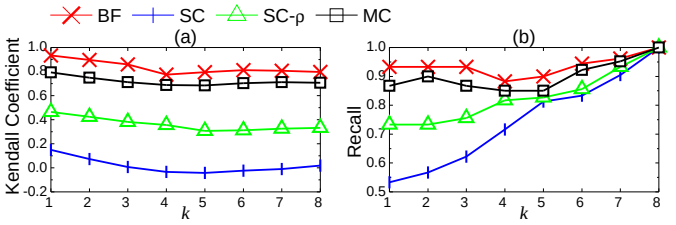


Figure 11. Effectiveness vs. k on Real Data

to BF, SC and SC-ρ are much poorer on both measures. Also, despite that MC has used a sufficient number of simulations for flow computing, there is still a gap between it and BF on the two measures. In our implementation, MC samples each positioning record by only considering the probabilistic samples in it, which

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2875096, IEEE Transactions on Knowledge and Data Engineering

ignores the information from contextual records and is inferior to keeping every probable sample and computing all valid paths as done by BF. Interestingly, as $|Q|$ is fixed in the tests, larger ks tend to include more ground truth query locations in the search result, and therefore both effectiveness measures improve after k is large enough. Eventually, all methods' recall increases to 1 as all locations in ground truth are returned to form the top-k results.

Next, fixing other parameters, we test $|Q|$ in different values. Referring to Figure 12(a), BF's τ decreases moderately with larger $|Q|$ as more query locations are involved in the search. However, it is still higher than 0.75 even though we do a complete query on all S-locations. Referring to Figure 12(b), BF's recall decreases as $|Q|$ becomes larger; but it is in general higher than 0.86 when $|Q|$ is increased to 80%. For both measures, BF outperforms other alternatives in all tests and decreases more slowly. The results show that our search is effective with large Qs on real data.

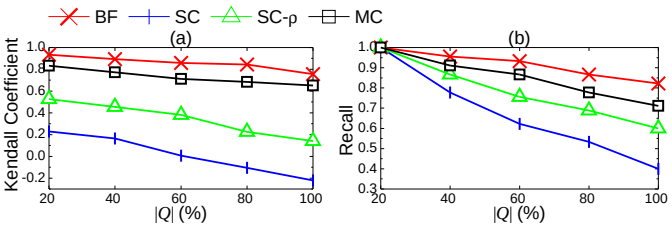


Figure 12. Effectiveness vs. $|Q|$ on Real Data

Last, we fix other parameters and vary Δt. Referring to Figure 13, both effectiveness measures of BF decrease with increasing Δt. However, they only decline slightly and their performance gaps with the other methods become larger as Δt increases. In all tests of BF, τ is higher than 0.82 and recall is higher than 0.88. When we extend the query time interval, objects' PSLs are enlarged, which involves more irrelevant S-locations in the flow computing. At the same time, a larger Δt tends to rule out more invalid paths and make the constructed paths more likely to approach the ground truth, which offsets the uncertainty in the mobility data and improves the accuracy of our flow computing. Due to these two conflicting reasons, BF's recall decreases only slightly when Δt increases. The same reasoning applies to the change of BF's Kendall coefficient. Therefore, our search is still effective when a long query time interval is used.

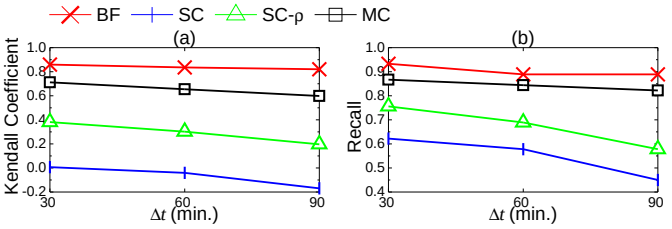


Figure 13. Effectiveness vs. Δt on Real Data

5.3 Experiments on Synthetic Data

We also evaluate our proposals on a very large synthetic dataset generated by the indoor data generator Vita [24]. We compare our methods with their alternatives in different settings, especially for $|O|$, T, and the indoor positioning error μ that may differ in other indoor mobility data.

Indoor Space and Locations. We duplicate a real-world floor plan [6] to generate a 5-floor building, where adjacent floors are connected by stairways and each floor takes $120m \times 120m$ with 100 rooms and 4 staircases. The irregular partitions in these entities are decomposed into smaller but regular ones, yielding 645 partitions and 840 doors in total. We convert each a staircase or a regular partition into an S-location. As a result, we have a total of 649 S-locations. All S-locations are indexed by an R-tree in which the root node contains 5 child nodes to distinguish different floors. We insert the four staircase S-locations into each floor's corresponding child node such that we can directly search them. The whole tree is around 5.9 MB and kept in main memory. On the other hand, the P-locations correspond to the reference points pre-selected for fingerprinting positioning algorithm [14]. We divide the entire indoor space using a grid, exclude those lattice points on the walls or outside the space, and use the remaining 5450 lattice points as P-locations. Among them, there are 4690 presence P-locations and 760 partitioning P-locations. Our generated S-locations and P-locations are used to build the two data structures introduced in Section 3.1. The total memory consumption of the data structures is up to 3.63 MB.

Moving Objects and IUPT. We generate indoor moving objects in the 5-floor building for a lifespan of two hours. Specifically, 2.5K (5K, 7.5K, or 10K) objects are randomly distributed to the floors, each having a lifespan varied from 30 minutes to 2 hours. The maximum speed for all objects is $V_{max} = 1m/s$. Object movement in a floor follows the random waypoint model [19]. In particular, an object moves towards its destination along the shortest indoor path, it stays in the destination for a random period of time from 5 to 30 minutes after arrival, and then it moves again to its next destination that is decided at random. For the entire simulation, we record object's exact location every second as its spatiotemporal trajectory. The trajectories with exact locations and timestamps form the ground truth in our experiments.

The synthetic IUPT is maintained according to the ground truth trajectories as follows. After an object has sent an update to IUPT, it keeps silent for at most T seconds, where T corresponds to the maximum positioning period defined in Section 5.2. An update for object o consists of a timestamp t and a sample set X, and $|X|$ is random between 1 and mss. We set $mss = 4$ in the tests. Following a typical model WkNN [14] used in fingerprinting, each sample in X is a pair of P-location loc_i and probability $prob_i$. In particular, loc_i is randomly within μ meters from o's current ground truth location $o.loc$, where μ denotes the *indoor positioning error*; and $prob_i = w(loc_i)/\sum_{1 \le k \le |X|} w(loc_k)$, where $w(loc_i)$ is the weight of P-location loc_i and computed as $1/(\mathrm{Dist}(loc_i, o.loc) \cdot (1+\gamma))$, meaning the weight is inversely proportional to the distance between loc_i and $o.loc$. We introduce randomness in computing $w(loc_i)$ by a random variable γ varying from -0.2 to 0.2.

Other Settings. The query locations in Q are randomly picked from S-location set $\mathcal{L}_S$. We vary the parameters according to Table 6 in which default values are given in bold. We compare the search methods BF (NL), SC, SC-ρ and MC. We set $\rho = 0.2$ in SC-ρ and simulation rounds to 25,000 in MC for an optimized

Table 6
Parameter Settings on Synthetic Data

Parameters	Settings
k	5, **10**, 15, 20
$\|Q\|$ (% of S-locations in $\mathcal{L}_S$)	**4**%, 8%, 12%
$\|O\|$	2.5K, **5K**, 7.5K, 10K
T (second)	1, 3, **5**, 7
μ (meter)	3, **5**, 7
Δt (minute)	15, **30**, 60, 120

6. https://goo.gl/wsdYys

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2875096, IEEE Transactions on Knowledge and Data Engineering

tuning in the default parameter setting. We randomly issue 20 queries in each setting and report the average measures.

5.3.1 Effect of Uncertainty in Synthetic Data

In addition to the mss studied in Section 5.2.2, we also investigate the effect of data uncertainty related to the maximum positioning period T and the indoor positioning error μ described above. Each of them is tested with other parameters fixed as default.

Effect of T. Referring to Figure 14(a), NL and BF's running time decreases clearly when we increase T from 1s to 7s. A smaller T means that the objects report their updates more frequently, and therefore more valid possible paths are to be processed in NL and BF. Nevertheless, BF can finish a half-hour query within 99s even all objects update their locations every second. Besides, the time costs of SC and SC-ρ decrease slightly as they are already minor. MC also decreases with an increasing T but much slower as its cost is only linearly correlated with the number of location reports. In each test, MC's cost is larger than the others by orders of magnitude since it has to use a large number of simulations to achieve a good enough effectiveness as reported below.

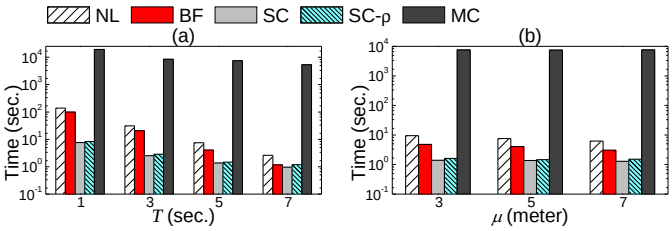


Figure 14. Efficiency vs. T and μ on Synthetic Data

As shown in Figure 15, when increasing T, both effectiveness measures decrease very rapidly for SC and SC-ρ but only slightly for BF and MC. A larger T makes the location updates less frequent, and therefore less information is captured in the IUPT, which causes the data uncertainty to increase and the query result quality to degrade. Nevertheless, BF still outperforms the other methods in the tests; its τ keeps above 0.77 for all T values and recall is higher than 0.9 in the default setting.

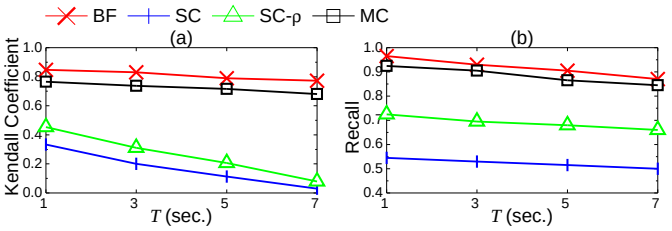


Figure 15. Effectiveness vs. T on Synthetic Data

Effect of μ. Next, we increase the positioning error in the data by varying μ from 3 meters to 7 meters. Referring to Figure 14(b), the running time of NL and BF decreases with increasing μ while that of the others is almost stable. When the positioning error becomes larger, the number of valid paths constructed by NL and BF tends to decline as our flow computing method can filter out more invalid paths generated from the inaccurate positioning results.

Accordingly, we report these methods' effectiveness results in Figure 16. When μ increases, both measures for SC and SC-ρ decrease clearly as these methods counting on the positioning records are very sensitive to the positioning errors in the data. In contrast, BF and MC perform much better as they make use of indoor topology to generate possible paths in flow computing. Still, BF outperforms MC because it considers the valid possible paths thoroughly. When $\mu = 7m$, its τ is still higher than 0.77 and its recall is over 0.87.

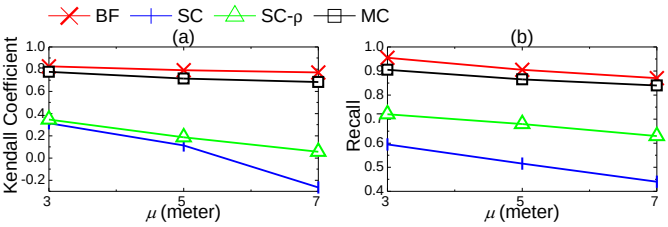


Figure 16. Effectiveness vs. μ on Synthetic Data

To sum up, those results on varying mss, T and μ altogether verify that our top-k search algorithm BF can work both efficiently and effectively even though the indoor mobility data is of relatively low quality.

5.3.2 Top-k Search Efficiency

Here we omit the efficiency results for k, $|Q|$, Δt that exhibit similar trends with the counterparts of Section 5.2.3, and focus on the efficiency performance on varying $|O|$ from 2.5K to 10K. Referring to Figure 17, it is easy to see that more moving objects result in longer running time in each method. Still, MC needs significantly more time than the others. When O contains 10K objects, NL takes around 15.2s to return the top-k results, while BF only requires 8.1s with a higher pruning capability. Although the simple counting methods are slightly faster, their effectiveness is considerably poorer than BF, as to be shown in Section 5.3.3.

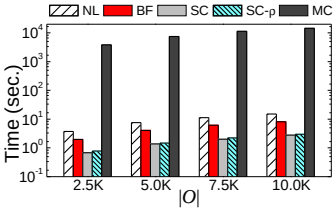


Figure 17. Efficiency vs. $|O|$ on Synthetic Data

5.3.3 Top-k Result Effectiveness

This section presents the effectiveness results on varying a parameter relevant to the top-k search and using the defaults for others.

Effect of k. Referring to Figure 18(a), BF's Kendall coefficient τ is above 0.93 for k up to 5, showing that its top-5 results are in good accord with ground truth. Afterwards, τ degrades since more query locations need to be computed and sorted. Nevertheless, it turns to improve when k grows to a certain number. Sufficiently large ks tend to include more ground truth in search result since $|Q|$ is fixed in the setting. Overall, BF's τ is higher than 0.77 and always outperforms MC in the tests. Referring to Figure 18(b), the recall of BF and MC decreases when increasing k, but BF's recall is still higher than that of MC and is above 0.89 for different k values. Note that both effectiveness measures of SC and SC-ρ are very low compared to those of BF and MC. This phenomenon can still be seen in the experimental results presented below.

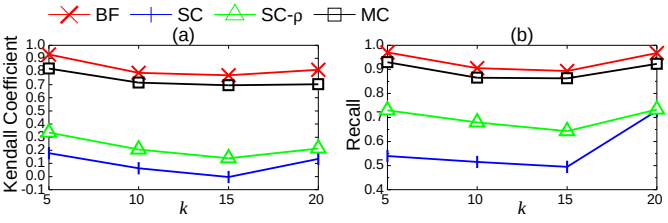


Figure 18. Effectiveness vs. k on Synthetic Data

Effect of $|Q|$. Referring to Figure 19, for all the methods, the two measures decrease as $|Q|$ increases because more query locations

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2875096, IEEE Transactions on Knowledge and Data Engineering

need to be considered. However, BF decreases more slowly than MC; its τ is still higher than 0.74 and its recall is above 0.83 when $|Q|$ increases to 12%.

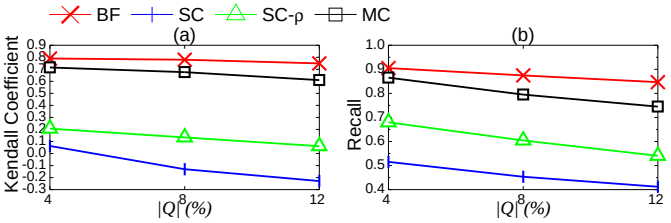


Figure 19. Effectiveness vs. $|Q|$ on Synthetic Data

Effect of $|O|$. Referring to Figure 20, the two measures of each method are only slightly affected by varying $|O|$. Besides, BF can always perform the best and both its measures stay very high, showing that our search is still effective for large object workloads.

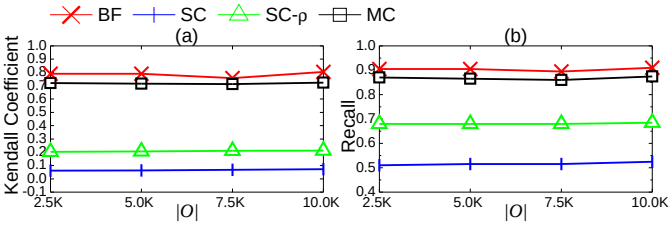


Figure 20. Effectiveness vs. $|O|$ on Synthetic Data

Effect of Δt. Referring to Figure 21, both measures decrease as Δt increases in each method, but BF still outperforms MC. As discussed in Section 5.2.4, larger Δts tend to produce qualified paths and thus improve the accuracy of flow computing; but larger Δt also tends to expand the objects' PSLs and thus deteriorates the accuracy. As a result, both measures of our search decline slightly when Δt is increased. These results verify that our method BF can still effectively process the queries with large Δt.

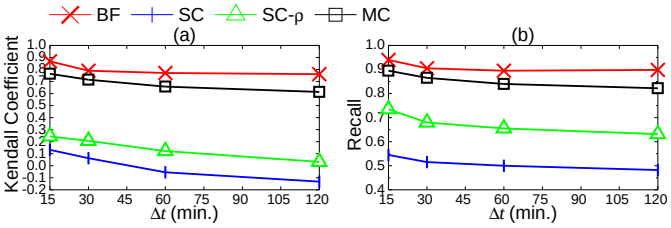


Figure 21. Effectiveness vs. Δt on Synthetic Data

Comparison with Alternatives. We compare our method with two indoor flow computing methods [3], [4], [27] using RFID data. One method [3], [4] is specialized for semi-constrained indoor movement environments (e.g., convey belt systems) where each semantic location features one entry and one exit, both having an RFID reader. Such a strict setting enables to count objects in a location within a past time interval. We use semi-constrained counting (SCC) to denote this method. The other method [27] uses uncertainty regions to capture an object's possible locations within a time interval, and computes the flow for an indoor location by summing up its intersection with each object's uncertainty region. We use UR to denote this method. Unlike SCC, both our method and UR work for general indoor environments. Using the generator Vita [24], we build an RFID tracking model and generate the corresponding tracking records[7] according to the same set of object trajectories underlying our synthetic data. We deploy ordinary RFID readers with 3-meter detection range [41] at doors. Following the experimental settings in [4], [27], reader detection ranges do not overlap in our setting. As a result, some doors are associated with no reader. Under these constraints, we maximize the number of readers and deploy 420 readers in total.

We vary k and $|Q|$ and report Kendall coefficient measures in Table 7. Overall, UR results in the lowest measure. It models object uncertainty regions by a series of ellipses whose two foci are decided by two readers, tending to add flows to S-locations close to the ground truth S-location. The tendency is reinforced when all readers are placed at doors, which always results in large ellipses. When $|Q| = 4\%$, SCC and BF achieve comparable results. However, SCC's τ deteriorates very rapidly when Q is larger. SCC's counting falls short when some doors have no readers, and the effect becomes more visible when more queries include more S-locations in ranking. Compared to SCC and UR, our method returns the top-k results considerably consistent with ground truth. We discuss their technical difference in Section 6.

Table 7
Kendall Coefficient Comparsion with RFID Tracking Methods

k	$\|Q\| = 4\%$			$\|Q\| = 8\%$			$\|Q\| = 12\%$		
	SCC	UR	BF	SCC	UR	BF	SCC	UR	BF
5	0.89	0.42	**0.93**	0.85	0.33	**0.87**	0.68	0.20	**0.80**
10	**0.80**	0.31	0.79	0.75	0.22	**0.78**	0.64	0.17	**0.75**
15	**0.78**	0.26	0.77	0.73	0.15	**0.76**	0.60	0.13	**0.74**
20	0.77	0.25	**0.81**	0.73	0.21	**0.75**	0.57	0.13	**0.74**

6 Related Work

Querying Indoor Space Moving Objects. In the context of RFID-type symbolic indoor tracking, Yang et al. study continuous range monitoring queries [39] and probabilistic k nearest neighbor queries [40]. Uncertain query results are returned as objects' locations are unknown when they are outside any RFID reader's detection range. To improve the query result quality, Yu et al. [41] use a particle filter to infer undetected object locations. Assuming a probabilistic sample based location data format, Xie et al. [36], [37] define indoor distance-aware spatial queries and design query processing algorithms. Unlike these works that query online indoor moving objects, the paper analyzes historical data.

Jensen et al. [18] study historical indoor object trajectories and propose 2D R-tree variants on RFID readers and timestamps to facilitate processing of specialized queries. Delafontaine et al. [10] analyze the moving patterns within historical Bluetooth tracking data. Given a past time or a time interval, Lu et al. define spatio-temporal joins [28] to find moving object pairs in the same indoor partition, and top-k queries [27] to find the most frequently visited indoor POIs. Ahmed et al. [3], [4] define threshold density query to find dense indoor semantic locations in a historical time interval. The paper distinguishes from the existing works on historical data in several aspects. First, unlike all these works, the paper aims at indoor positioning data captured as probabilistic samples. Second, the paper defines flows for indoor regions based on the uncertain location samples produced in general indoor environments, whereas the simple counting method [3], [4] is restricted to semi-constrained indoor settings. Third, our indoor flow computing makes full use of multiple probabilistic samples in capturing an object's actual movements, while work [27] employs rigid circular ranges to derive object uncertainty regions that tend to be too large. Consequently, the techniques in [3], [4], [27] fall short in solving the paper's problem.

7. A record (o, r_i, t_s, t_e) means that object o is in reader r_i's range from time t_s to t_e.

Indoor Semantic Location Extraction. Kang et al. [20] propose clustering techniques to extract significant locations from coordinate points captured by Wi-Fi positioning. PlaceSense [21] identifies semantically meaningful places from Wi-Fi AP fingerprints. Eigenplaces [7] segments a space into different places that feature different Wi-Fi usage characteristics. Chen et al. [9] employs classification to extract indoor semantics locations with different labels. Unlike this works, our study finds the top-k locations with the highest flows from a set of pre-known semantic locations.

Flow Analysis in Outdoor Setting. Assuming Euclidean spaces, Tao et al. [31] propose techniques to count spatio-temporal objects within a given spatial window during a given time interval. Cao et al. [8] propose methods for identifying top-k significant locations semantically from GPS data. Xue et al. [38] study the destination prediction problem in a sparse data setting. Our work is clearly different from these works. First, our work utilizes indoor topology to support our search on discrete indoor mobility data, whereas the others do not support indoor topology and are inapplicable to computing the indoor flows. Second, to model the object movements, our work constructs the possible paths from uncertain indoor positioning records rather than the certain GPS sequences studied in [8], [31]. Third, our work computes a possible path's probability based on the probabilistic samples reported at each timestamp, whereas work [38] uses those transition probabilities between the destinations that are learned from the historical data.

Wei et al. [33] conduct path inference and produce an uncertain trajectory by cross-referring to other trajectories on similar routes. Zheng et al. [42] complement uncertainty sections of road network trajectories and plan the hottest routes for a query location sequence. Su et al. [30] design an anchor-based calibration method that aligns trajectories to a set of fixed anchor points in path inference from historical trajectories. Our work also clearly differs from these studies. First, unlike these works that only focus on low-sampling data uncertainty, our work also considers the positioning uncertainty in the context of complex indoor topology. Second, our work studies on indoor spaces, where the movement constraints [26] are modeled differently from free space [33] or road network [30], [42]. Third, our work uses indoor topology to enable reliable flow computing, while the other works reduce the uncertainty by referring to other historical trajectories.

7 Conclusion And Future Work

This paper tackles the problem of finding top-k popular indoor semantic locations from indoor mobility data captured as probabilistic samples. We formulate a reliable indoor flow definition by considering both data uncertainty and indoor topology. We design a complete set of techniques to enable efficient indoor flow computing, and search algorithms for finding the top-k popular semantic locations. The experimental studies on real and synthetic data verify that our flow computing techniques work efficiently, and our search algorithms are efficient, scalable and effective.

For future work, it is interesting to model object behaviors in order to further improve the measuring and quantifying of indoor flows. Also, it is possible to study historical densities for indoor locations by considering the impact of their sizes. Furthermore, it is relevant to consider an online and continuous version of the top-k popular location query in similar scenarios.

References

[1] InLocation Alliance. http://www.in-location-alliance.com/.

[2] List of countries by smartphone penetration. http://goo.gl/pdtvMM.

[3] T. Ahmed, T. B. Pedersen, and H. Lu. Finding dense locations in indoor tracking data. In *MDM*, pp. 189–194, 2014.

[4] T. Ahmed, T. B. Pedersen, and H. Lu. Finding dense locations in symbolic indoor tracking data: modeling, indexing, and processing. *GeoInformatica*, 21(1): 119–150, 2017.

[5] T. Becker, C. Nagel, and T. H. Kolbe. A multilayered space-event model for navigation in indoor spaces. *3D Geo-Info*, pp. 61–77, 2009.

[6] R. Broberg and F. Gadnell. Platform-independent indoor positioning system. Master's thesis, Uppsala University, 2013.

[7] F. Calabrese, J. Reades, and C. Ratti. Eigenplaces: segmenting space through digital signatures. *IEEE Pervasive Computing*, 9(1): 78–84, 2010.

[8] X. Cao, G. Cong, and C. S. Jensen. Mining significant semantic locations from GPS data. *PVLDB*, 3(1): 1009–1020, 2010.

[9] Z. Chen, Y. Chen, S. Wang, and Z. Zhao. A supervised learning based semantic location extraction method using mobile phone data. In *CSAE*, 3(1): 548–551, 2012.

[10] M. Delafontaine, M. Versichele, T. Neutens, and N. Van de Weghe. Analysing spatiotemporal sequences in Bluetooth tracking data. *Applied Geography*, 34: 659–668, 2012.

[11] G. Deak, K. Curran, and J. Condell. A survey of active and passive indoor localisation systems. *Computer Communications*, 35(16): 1939–1954, 2012.

[12] I. Hwang and Y. J. Jang. Process mining to discover shoppers pathways at a fashion retail store using a WiFi-base indoor positioning system. *IEEE Trans. Auton. Sci. Eng.*, 14(4): 1786–1792, 2017.

[13] S. Hwang, K. Kwon, S. K. Cha, and B. S. Lee. Performance evaluation of main-memory R-tree variants. In *SSTD*, pp. 10–27, 2003.

[14] V. Honkavirta, T. Perala, S. Ali-Loytty, and R. Piché. A comparative survey of WLAN location fingerprinting methods. In *WPNC*, pp. 243–251, 2009.

[15] J. Hightower and G. Borriello. Location systems for bbiquitous computing. *IEEE Computer*, 34(8): 57–66, 2001.

[16] P. L. Jenkins, T. J. Phillips, E. J. Mulberg, and S. P. Hui. Activity patterns of californians: use of and proximity to indoor pollutant sources. *Atmospheric Environment*, 26(12), 1992.

[17] C. S. Jensen, H. Lu, and B. Yang. Graph model based indoor tracking. In *MDM*, pp. 122–131, 2009.

[18] C. S. Jensen, H. Lu, and B. Yang. Indexing the trajectories of moving objects in symbolic indoor space. In *SSTD*, pp. 208–227, 2009.

[19] D. B. Johnson and D. A. Maltz. Dynamic source routing in ad hoc wireless networks. In *Mobile computing*, pp. 153–181, 1996.

[20] J H. Kang, W. Welbourne, B. Stewart, and G. Borriello. Extracting places from traces of locations. In *WMASH*, pp. 110–118, 2004.

[21] D H. Kim, J. Hightower, R. Govindan, and D. Estrin. Discovering semantically meaningful places from pervasive RF-beacons. In *Ubicomp*, pp. 21–30, 2009.

[22] N. Klepeis, W. Nelson, W. Ott, J. Robinson, A. Tsang, P. Switzer, J. Behar, S. Hern, and W. Engelmann. The National Human Activity Pattern Survey (NHAPS): a resource for assessing exposure to environmental pollutants. *Journal of Exposure Science*, 11(3): 231, 2001.

[23] J. Lee. A spatial access-oriented implementation of a 3-D GIS topological data model for urban entities. *GeoInformatica*, 8(3), 2004.

[24] H. Li, H. Lu, X. Chen, G. Chen, K. Chen, and L. Shou. Vita: A versatile toolkit for generating indoor mobility data for real-world buildings. *PVLDB*, 9(13): 1453–1456, 2016.

[25] H. Liu, H. Darabi, P. Banerjee, and J. Liu. Survey of wireless indoor positioning techniques and systems. *IEEE Trans. Knowl. Systems, Man, Cybernetics*, 37(6), 2006.

[26] H. Lu, X. Cao, and C. S. Jensen. A foundation for efficient indoor distance-aware query processing. In *ICDE*, pp. 438–449, 2012.

[27] H. Lu, C. Guo, B. Yang, and C. S. Jensen. Finding frequently visited indoor POIs using symbolic indoor tracking data. In *EDBT*, pp. 449–460, 2016.

[28] H. Lu, B. Yang, and C. S. Jensen. Spatio-temporal joins on symbolic indoor tracking data. In *ICDE*, pp. 816–827, 2011.

[29] C. F. Ng. Satisfying shoppers psychological needs: From public market to cybermall. *Journal of Environmental Psychology*, 23(4): 439–455, 2013.

[30] H. Su, K. Zheng, H. Wang, J. Huang, and X. Zhou. Calibrating trajectory data for similarity-based analysis. In *SIGMOD*, pp. 833–844, 2013.

[31] Y. Tao, G. Kollios, J. Considine, F. Li, and D. Papadias. Spatio temporal aggregation using sketches. In *ICDE*, pp. 214–225, 2004.

[32] Y. Tao and D. Papadias. Range aggregate processing in spatial databases. *IEEE Trans. Knowl. Data Eng.*, 16(12): 1555–1570, 2004.

[33] L.-Y. Wei, Y. Zheng, and W.-C. Peng. Constructing popular routes from uncertain trajectories. In *KDD*, pp. 195–203, 2012.

[34] E. Whiting, J. Battat, and S. Teller. Topology of Urban Environments. *CAADFutures*, pp. 114–128, 2007.

[35] M.F. Worboys. Modeling indoor space. In *ISA*, pp. 1–6, 2011.

[36] X. Xie, H. Lu, and T. B. Pedersen. Efficient distance-aware query evaluation on indoor moving objects. In *ICDE*, pp. 434–445, 2013.

[37] X. Xie, H. Lu, and T. B. Pedersen. Distance-aware join for indoor moving objects. *IEEE Trans. Knowl. Data Eng.*, 27(2): 428–442, 2015.

[38] A. Y. Xue, J. Qi, X. Xie, R. Zhang, J. Huang, and Y. Li. Solving the data sparsity problem in destination prediction. *The VLDB Journal*, 24(2): 219–243, 2015.

[39] B. Yang, H. Lu, and C. S. Jensen. Scalable continuous range monitoring of moving objects in symbolic indoor space. In *CIKM*, pp. 671–680, 2009.

[40] B. Yang, H. Lu, and C. S. Jensen. Probabilistic threshold k nearest neighbor queries over moving objects in symbolic indoor space. In *EDBT*, pp. 335–346, 2010.

[41] J. Yu, W.-S. Ku, M.-T. Sun, and H. Lu. An RFID and particle filter-based spatial query evaluation system. In *EDBT*, pp. 263–274, 2013.

[42] K. Zheng, Y. Zheng, X. Xie, and X. Zhou. Reducing uncertainty of low-sampling-rate trajectories. In *ICDE*, pp. 1144–1155, 2012.